Lecture 5

Inf1A: Limits of Finite State Machines

5.1 Introduction

In this lecture we will look at the limitations of Finite State Machines. We will see that the most appealing property of Finite State Machines (their simplicity) has a limiting effect on the types of computations that can be performed by an FSM.

We will give an example of a natural language, the language consisting of all *palindromes* over the alphabet Σ, which nevertheless cannot be accepted by any FSM. Therefore it is not a regular language, and (using what we know from Lecture 4) it cannot be represented by *any* regular expression. There are many other examples of languages which cannot be accepted by any FSM. As a rule, the languages which cannot be accepted by FSMs tend to be languages which require a "count" of a possibly unlimited value (or a "record" of possibly unlimited size) to be maintained.

We will give a very gentle introduction to another class of languages called *context-free languages*, which are not in general accepted by Finite State Machines. We will show that because this class of languages is more powerful, the set of palindromes can be represented by a *context-free grammar*.

5.2 "Palindromes" is not a regular language

Let Σ be any alphabet. A palindrome is any string (or "word") over that alphabet that reads the same either forwards or backwards. For example, in the English language, the strings RADAR, KAYAK, NOON and EYE are all palindromes.

If we take the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, some examples of palindromes in Σ^* are

$$\epsilon, \mathtt{a}, \mathtt{b}, \mathtt{aa}, \mathtt{bb}, \mathtt{aaa}, \mathtt{aba}, \ldots$$

Now we will prove a theorem that tells us that there is no Finite State Machine that recognises the set of palindromes over the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. Although we work with $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, the proof will go through for any alphabet which has at least two symbols. It's a nice exercise to check this!

Theorem 5.1: *There is no Finite State Machine to recognise the set of palindromes over the alphabet* $\{\mathsf{a}, \mathsf{b}\}$*. Therefore the set of palindromes over* $\{\mathsf{a}, \mathsf{b}\}$ *is not a regular language.*

Proof: We prove this Theorem using the technique of *proof by contradiction*. That means that we assume the opposite of what we want to prove, and make a series of logical deductions from our assumption. If we eventually end up contradicting ourselves, then we know that our assumption is not consistent with what we know, and therefore it must be false.

So, suppose that the Theorem is false. Well then there must be *some* Finite State Machine that accepts the language of palindromes over $\{\mathsf{a}\,\mathsf{b}\}$. Also, because every Non-deterministic Finite State Machine can be converted into an equivalent Deterministic Finite state Machine (to accept the same language), therefore in fact there is some deterministic Finite State Machine that accepts the language of palindromes over $\{\mathsf{a}, \mathsf{b}\}$. Let this deterministic FSM be called M.

Now, M is a FSM, which means that it must have a *finite* number of states. We should not make any assumptions about how many states M has (well, we can assume that M has at least 2 states), so instead let us use the symbol k to represent the (unknown) finite number of states of M. Now let us use a trick, working with this unknown number of states k.

Think about the string $\mathsf{a}^k\mathsf{b}^k\mathsf{a}^k$, where x^k means "x repeated k times". Now clearly this string is a palindrome, because it reads the same both backwards and forwards. Therefore, since M is a machine accepting the language of palindromes over $\{\mathsf{a}, \mathsf{b}\}$, it *must* accept the string $\mathsf{a}^k\mathsf{b}^k\mathsf{a}^k$.

Now think about the *trace* that the string $\mathsf{a}^k\mathsf{b}^k\mathsf{a}^k$ takes through the FSM M (because our string is accepted, and because M is deterministic, it must have exactly one trace). Remember for a D-FSM, that whenever a computation takes a transition in the D-FSM, exactly one input symbol is read. When we start reading $\mathsf{a}^k\mathsf{b}^k\mathsf{a}^k$, we start at the start state s_0 of our unknown machine M. As we read each input symbol, we take a transition. The trace of the machine will look like

$$s_0,\ \mathsf{a},\ s_1,\ \ldots,\ \mathsf{a},\ s_k,\ \mathsf{b},\ s_{k+1},\ \ldots,\ \mathsf{b},\ s_{2k},\ \mathsf{a},\ s_{2k+1},\ \ldots,\ \mathsf{a},\ s_{3k},$$

where each of the s_i are states of the machine M (though not all different states). Notice that in reading just the first k symbols of our string, which are all as, we take k transitions. The list of states visited while reading the first k symbols of our string, is $s_0, s_1, \ldots, s_k$. Now this list has $k+1$ elements, but there are only k states in the machine M in total. So that means that in the list $s_0, s_1, \ldots, s_k$ (which has length $k+1$), there must be *some* state from Ms state set which occurs more than once. Also, since M is deterministic, every transition consumes one input symbol, so the machine reads *at least one* a in between two visits to this state.

Let the state that is revisited be called q[1] and let i (for some value $1 \leq i \leq k$) be the number of as read in between two visits to q (we are not going to try to make any guesses about the exact number of as that are read). Also, let ℓ be the number of as that are read before the first visit to q. Since the string $\mathsf{a}^k\mathsf{b}^k\mathsf{a}^k$ is accepted by the FSM starting at s_0, the string $\mathsf{a}^{k-\ell}\mathsf{b}^k\mathsf{a}^k$ is accepted by the FSM when starting at q.

There is a diagrammatic representation of the section in Figure 5.1 below (remember

[1] By the way, if there are two or more states which are revisited, just choose the first of these states (that the computation meets) for q.

that the loop from q to itself may pass through many states on its return to q, even though they are not represented).

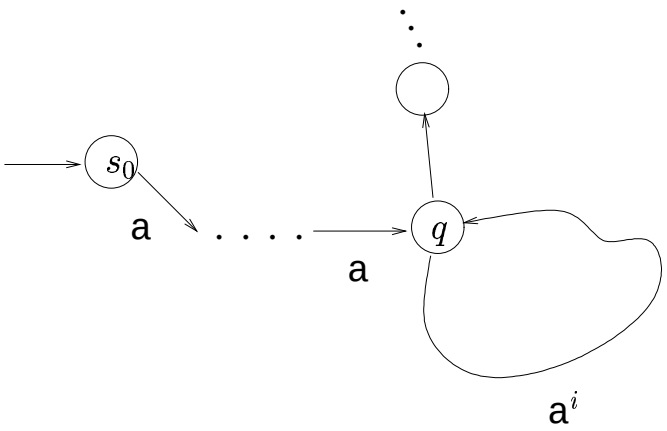

Figure 5.1: Diagram of the "loop" that we encounter in reading a^k

Now consider the new string $\mathsf{a}^{k+i}\mathsf{b}^k\mathsf{a}^k$. Think about what happens when this string is input to M. Well, the trace for reading the first ℓ as leads to the state q, by definition of ℓ above. Then the remaining string to be read (starting at q) is $\mathsf{a}^{k+i-\ell}\mathsf{b}^k\mathsf{a}^k$. Now remember that if we read i as starting at state q, we will return to q. Then the remaining string to be read (starting again from q) is $\mathsf{a}^{k-\ell}\mathsf{b}^k\mathsf{a}^k$. However, we already know (by our analysis of string $\mathsf{a}^k\mathsf{b}^k\mathsf{a}^k$ above) that the string $\mathsf{a}^{k-\ell}\mathsf{b}^k\mathsf{a}^k$ will be accepted when we start reading this string from q.

Therefore, the machine M will accept $\mathsf{a}^{k+i}\mathsf{b}^k\mathsf{a}^k$ also. However this string is *not* a palindrome! This is a contradiction. Therefore our assumption that there exists an FSM M to accept the set of strings over $\{\mathsf{a}, \mathsf{b}\}$ which are palindromes must have been incorrect. □

The idea of proof by contradiction is difficult to get used to when it is new to you. In doing these proofs, the most important thing is that you have to make sure that apart from your main assumption (which is the opposite of what you want to prove) you must make no other assumption at all, and only make deductions which are rigorously true. Then at the end, when you do get a contradiction, there is only one thing that could have caused this contradiction, your original assumption. Hence that assumption must be false (and you have proven your real goal).

5.3 Context-free Grammars and Context-free Languages

5.3.1 Non-regular languages

Theorem 5.1 above shows that there are some languages (our example was the language of palindromes) which are *not* regular - that is, they do not correspond to the language accepted by *any* FSM, and equivalently, there is no regular expression to represent the language.

The language of palindromes is not just a freak exception, as there are many other languages which cannot be accepted by any FSM. Another example of a language which

cannot be accepted by any FSM (or equivalently, cannot be represented by any r.ex.) is

$$L = \{\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N}\}.$$

In general, non-regular languages tend to be those languages which require an arbitrarily large "count" (or "record" of the part of the string which has been seen) to be maintained during the process of reading the string. This is not possible with a Finite State Machine, because it only has a limited finite amount of storage available.

5.3.2 Context-free Languages

However, there are other formal representations for specifying languages apart from regular expressions, and some of these allow us to specify some non-regular languages. One such example is the notion of a *Context-Free Grammars*. We are not going to study these in detail, but we will give a taste. Here is the formal definition.

Definition 5.2: *A* Context-free Grammar *is defined by a* 4*-tuple* (V, Σ, S, P)*, where* V *is a set of* non-terminal *(or variable) symbols,* Σ *is a an alphabet of* terminal *symbols, where* $V \cap \Sigma = \emptyset$*, where* $S \in V$ *is the* start symbol*, and* P *is a set of* production rules *of the form* $X \to \alpha$*, for* $X \in V$ *and* $\alpha \in (V \cup \Sigma)^*$*.*

Definition 5.3: *Let* (V, Σ, S, P) *be a Context-free Grammar.*

(i) Let $w, w' \in (V \cup \Sigma)^*$*. We say that* $w \Rightarrow w'$ *if and only if there is some production rule* $X \to \alpha \in P$ *such that we can obtain* w' *by replacing some instance of the variable* X *in* w *by the expression* α*.*

(ii) We say that $w \Rightarrow^* w'$ *if and only if there is some* $n \in \mathbb{N}$*, and some sequence* $w_0 = w, w_1 \ldots, w_n = w'$ *such that* $w_i \Rightarrow w_{i+1}$ *for* every $i = 0, \ldots, n-1$*. (when* $n = 0$ *we have* $w = w'$*). The sequence*

$$w \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w'$$

is called a derivation *of the grammar.*

(iii) The language $L(G)$ *is defined as the following set of strings of terminals:*

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}.$$

We say that a language L is a *Context-free Language* if and only if there is some Context-free Grammar G such that $L = L(G)$.

In general, we say that a string $w \in (V \cup \Sigma)^*$ (which can contain both terminals and non-terminals) is *generated* by G if $S \Rightarrow^* w$.

The definition looks quite formal, but in practice context-free grammars are quite easy to work with. On the next page we give a context-free grammar to generate the set of palindromes over the alphabet $\{\mathtt{a}, \mathtt{b}\}$.

We define $G_{palin} = (V, \Sigma, S, P)$ as follows:
Let $V = \{S\}$.
Let $\Sigma = \{\mathsf{a}, \mathsf{b}\}$.
The start symbol is S.
The set of production rules P is defined as the following set of rules: [2]

$$\begin{array}{rcl} \{S & \rightarrow & \mathsf{a}S\mathsf{a} \\ S & \rightarrow & \mathsf{b}S\mathsf{b} \\ S & \rightarrow & \epsilon \mid \mathsf{a} \mid \mathsf{b}\} \end{array}$$

First we will give an example of a *derivation* of an element of $L(G_{palin})$:

$$S \Rightarrow \mathsf{b}S\mathsf{b} \Rightarrow \mathsf{ba}S\mathsf{ab} \Rightarrow \mathsf{baaab}.$$

Notice that the first step of the derivation is obtained by using the second production rule, the second step by using the first derivation rule, and the final step by using one of the options of the third derivation rule.

Now clearly, baaab is a palindrome. Also, it is probably clear to you that any string of terminals (ie, of as and bs) generated by G_{palin} will be a palindrome. We need to argue in the opposite direction. Is it true that *every* palindrome over $\{\mathsf{a}, \mathsf{b}\}$ can be generated by the grammar G_{palin}?

I argue now that the reason is yes. Consider an arbitrary palindrome over $\{\mathsf{a}, \mathsf{b}\}$. There are three cases to consider - the empty string ϵ, a non-empty string of even length, and a string of odd length. In the empty string case, we have an immediate production rule which we can use to derive ϵ in one step. In the case where the palindrome is non-empty, and the length is an even number $2k$, we simply list the first k elements of the palindrome, then apply the rules by following that list, applying the first production rule when we see an a and applying the second rule when we see a b. We finish with an application of the $S \rightarrow \epsilon$ rule. In the case where the palindrome has some odd length $2k + 1$, we again list the first k elements of the string, and perform a sequence of applications of the first and second rules, using the list to determine which rule should be applied at each step. Then we finish by checking the $k+1$th element of the palindrome, and by using either the rule $S \rightarrow \mathsf{a}$ or the rule $S \rightarrow \mathsf{b}$ appropriately.

This is not a formal proof (the best way to formally prove that $L(G_{palin})$ generates the language of palindromes is to use *proof by induction*). But I hope the general argument above is convincing.

As a general comment, you might have noticed that context-free grammars are *syntactic definitions*, like regular expressions. However, we know that the class of languages represented by regular expressions is equivalent to the class of languages accepted by Finite State Machines. So we have a computational view of regular expressions as well as the basic definition. In later years of the *Informatics* course you will learn that it is actually possible to relate context-free languages to the computational model of *pushdown automatas*.

As another general comment, there exist languages (called *context-sensitive languages*) which cannot be generated by any context-free grammar.

[2] The last production rule is equivalent to writing five rules $S \rightarrow \epsilon$, $S \rightarrow \mathsf{a}$ and $S \rightarrow \mathsf{b}$. It is common to use the | operator to shorten the list of production rules.

5.4 Online resources

1. For more information on context-free languages see `Wikipedia` at:

`http://en.wikipedia.org/wiki/Context-free_language`

2. Context-free grammars are commonly used to define the language of all *syntactically-correct programs* in a given language. When this is done, the format of the Grammar description is given in a slightly different form known as *Backus-Naur form*.

I had a look for *Backus-Naur form* descriptions of the `Java` programming language (which you will cover next semester) on the web.

There is an official description at the following webpage (but this is not really Backus-Naur form).

`http://java.sun.com/docs/books/jls/second_edition/html/syntax.doc.html`

There is a Backus-Naur definition of `Java` at the webpage given below:

`http://www.math.grin.edu/ stone/courses/languages/Java-syntax.xhtml`

It is nicer to read than the link above, even though it is not an official document.

These notes have been adapted from two sets of earlier notes, the first due to Stuart Anderson, Murray Cole and Paul Jackson, and the second due to Martin Grohe and Don Sannella.

Mary Cryan, November 2004